An element-by-element multilevel block ILU preconditioner using GLAS

N. Vannieuwenhoven and K. Meerbergen

Department of Computer Science, K.U.Leuven

Abstract. We investigate a new element-by-element multilevel block-ILU preconditioner that combines the advantages of element-by-element (EBE) and incomplete LU-factorization (ILU) preconditioners. The preconditioner is constructed in EBE fashion with high-throughput dense operations. Numerical experiments demonstrate its effectiveness. Speedups as high as five were obtained over ILU(0) with a C++ implementation using GLAS.

Keywords: element-by-element, incomplete LU-factorization, multilevel, preconditioner
PACS: 02.60.Dc, 02.60.Cb, 02.70.Dh

INTRODUCTION

In this short paper, we outline an efficient preconditioner to solve large-scale sparse linear systems $A\mathbf{x} = \mathbf{b}$, using Krylov-subspace iterative methods. We assume that the system admits an *element structure*:

$$A := \sum_{e \in E} A^{[e]} \qquad \text{and} \qquad \operatorname{Sp}\left(A^{[e]}\right) := I(e) \times I(e),$$

where E is a set of *elements* and $A^{[e]}$ are sparse *element matrices*. $\operatorname{Sp}(A)$ denotes the *sparsity* of A; that is, the set of positions at which A is non-zero. The (symmetric) *index set* $I(e)$ *defines* the sparsity of the element matrix of e. Two elements are called neighbours if their index sets overlap. In general, an element matrix does not correspond to a submatrix of the assembled system, due to overlapping index sets. Systems with an element structure arise frequently, e.g. from the finite element method.

In the context of this work, we aim to combine the features of the element-by-element and incomplete LU-factorization preconditioners. Element-by-element (EBE) preconditioners exploit the element structure of A. They operate exclusively on the element matrices. A is typically approximated by combining the LU-factorizations of the element matrices [1]. They are presently out of favor due to their limited effectiveness. Incomplete LU-factorization preconditioners are more generally applicable. These are effective on a variety of problems, including scalar elliptic partial differential equations (PDE) [2]. By operating on sparse matrices, they attain a more limited throughput.

In [3], we presented a new EBE multilevel block-ILU preconditioner, called EBE-ILU, for systems with an element structure. It constructs a block-ILU factorization, with multilevel structure, in an EBE fashion. It may be constructed and applied using high-throughput operations on dense matrices. EBE-ILU aims to offer a high-throughput solution phase while remaining as effective, in preconditioning quality, as ILU-type preconditioners. EBE-ILU is targeted at problems with large element matrices. In FE problems, these could result from higher-order elements or multiple unknowns per node. We also target problems in which multiple right-hand sides are concurrently solved for.

The outline of the paper is as follows. In the next section we present the high-level outline of the EBE-ILU preconditioner. Thereafter, we show numerical examples measuring the performance w.r.t. ILU(0). Finally, the conclusions are presented.

AN EBE-ILU PRECONDITIONER

In [3], a multilevel block ILU-factorization was proposed, inspired by BILUM [4]. At level k, A_k is approximately factored as

$$A'_k := P_k A_k P_k^T = \begin{bmatrix} D_k & U_k \\ L_k & S_k \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ L_k D_k^{-1} & I \end{bmatrix} \begin{bmatrix} D_k & U_k \\ 0 & A_{k+1} \end{bmatrix} := \hat{L}_k \hat{U}_k, \tag{1}$$

where $A_{k+1} \approx S_k - L_k D_k^{-1} U_k$ is called the *reduced system* and $A_1 := A$. The permutation matrix P_k is determined such that D_k forms a block diagonal matrix. The product $L_k D_k^{-1}$ is not computed. Rather, its factors are stored. D_k^{-1} is computed and stored. The factorization continues recursively on A_{k+1}.

It is assumed that A_k admits an element structure. In [3], that structure was exploited in the design of efficient algorithms to compute the multilevel factorization in Equation (1). The factorization algorithm operates on the element structure of A_k, rather than the sparse matrix A_k. It constructs a new element structure that represents the reduced system. Here, we briefly outline the element-by-element factorization algorithm detailed in [3].

The block diagonal matrix D_k is constructed by selecting a subset of elements $\Delta_k \subseteq E_k$, called the *diagonal elements*. The elements in this set should form a block independent set [4]. That is, there should be no edge between any two elements in the set of diagonal elements. The coefficient matrices of these elements are then modified in a structured manner, using a *local assembly* operation, in order to let them correspond to a proper submatrix of A_k. Given an element e, that consists of adding the overlapping submatrices of the elements neighbouring e to the coefficient matrix of e, while setting said submatrices to zero in the neighbouring elements. The operation does not alter the assembled system. The diagonal elements impose the permutation matrix P_k and represent the block diagonal matrix D_k. Consequently, the inverse of D_k may be computed EBE by inverting the element matrices of the diagonal elements.

The product $L_k D_k^{-1}$ is not computed but rather its factors are stored seperately. The matrix L_k is stored as a sparse matrix in compressed sparse column storage format [5].

To compute the reduced system A_{k+1}, we proceed as follows. It is demonstrated in [3] that the computation of the global Schur complement $S_k - L_k D_k^{-1} U_k$ may be decomposed in small local Schur complements. Each of these local Schur complements is uniquely associated with one diagonal element. In fact, for a given diagonal element $d \in \Delta_k$ the local Schur complement may be computed by adding the element coefficient matrices of d and its neighbours and computing the Schur complement of that matrix (where the indices of d determine the partitioning). The non-fill-in values of the local Schur complements are then *distributed* over the elemens in $E_k \backslash \Delta_k$. Distribution is the inverse of assembly. It consists of adding a value at (i, j) in the local Schur complement to exactly one element matrix that has that position in its sparsity. By distributing the non-fill-in values, we obtain an approximation to the global Schur complement. The reduced system is then given by that approximation. Its element structure consists of the elements in $E_{k+1} := E_k \backslash \Delta_k$.

To solve the multilevel system in Equation (1), we proposed a hybrid sparse-EBE technique. It combines the advantages of sparse and dense data structures. Sparse data structures have an efficient memory utilization and use the fewest operations for the sparse-matrix vector product. It has the disadvantage of inefficient indirect memory accesses. If a block structure is present, a block-based storage is more efficient. It may achieve higher throughput. EBE-ILU is well-suited to a hybrid approach. D_k^{-1} is stored as a sequence of (variable size) blocks. L_k and U_k are sparse and unstructured, in general. Compressed storage [5] is preferable. At every level k, the factored system is solved for a vector as follows. Assume that the unknowns are renumbered such that $P_k = I$ for all k. Considering Equation (1), we solve $\hat{L}_k \hat{U}_k \mathbf{x}_k = \mathbf{b}_k$, as

$$
\begin{aligned}
&1.\ \mathbf{z}_{k,1} := \mathbf{b}_{k,1}, && 3.\ \mathbf{x}_{k,2} = A_{k+1}^{-1} \mathbf{z}_{k,2}, \\
&2.\ \mathbf{z}_{k,2} := \mathbf{b}_{k,2} - L_k (D_k^{-1} \mathbf{z}_{k,1}), && 4.\ \mathbf{x}_{k,1} = D_k^{-1} (\mathbf{z}_{k,1} - U_k \mathbf{x}_{k,2}),
\end{aligned}
$$

where $\mathbf{x}_k := \begin{bmatrix} \mathbf{x}_{k,1}^T & \mathbf{x}_{k,2}^T \end{bmatrix}^T$, $\mathbf{b}_k := \begin{bmatrix} \mathbf{b}_{k,1}^T & \mathbf{b}_{k,2}^T \end{bmatrix}^T$ and $A_1 \mathbf{x}_1 = \mathbf{b}_1$ is the original system to solve. System 3 is solved recursively. D_k^{-1} is applied by employing a dense matrix-vector product for every block. If the system is solved for multiple right-hand sides concurrently, a dense matrix product is used.

NUMERICAL EXPERIMENTS

We compare the performance of EBE-ILU w.r.t. ILU(0) with RCM renumbering using BiCgSTAB [6] as accelerator on finite element problems. Both preconditioners discard all fill-in values and thus require the same amount of memory to store the numerical values. They were implemented in C++ using the GLAS [7] library. It provides the implementation of the sparse and dense matrices, the matrix-vector product and the accelerator.

The experiments were conducted on an Intel Core2Duo P7350 processing unit running at 2.0GHz with 3MB L2 cache and 4GB main memory (clocked at 1066MHz).

Computational performance. Here, we investigate the computational performance of the forward and backward substitution phase of EBE-ILU and compare it with ILU(0). The code was compiled with the Intel C++ 11.0 20090131

Table 1. Experiments on the throughput of EBE-ILU and ILU(0) on an artificial 2D grid problem.

ES	Nnz	Operation count			Execution time (ms)			Throughput (Mflop/s)		
		ILU	EBE	Diff	ILU	EBE	Diff	ILU	EBE	Diff
4	37249	74498	109906	+47.5%	0.212	0.429	+102.4%	351.4	256.2	−27.1%
8	148996	297992	422724	+41.9%	0.964	0.886	−8.1%	309.1	477.1	+54.4%
12	335241	670482	938454	+40.0%	2.300	1.941	−15.6%	291.5	483.5	+65.9%
16	595984	1191968	1657096	+39.0%	4.283	2.979	−30.4%	278.3	556.3	+99.9%
20	931225	1862450	2578650	+38.5%	7.205	5.307	−26.3%	258.5	485.9	+88.0%
24	1340964	2681928	3703116	+38.1%	10.112	7.179	−29.0%	265.2	515.8	+94.5%

Table 2. Experiments on the throughput of EBE-ILU and ILU(0) on an artificial 3D grid problem.

ES	Nnz	Operation count			Execution time (ms)			Throughput (Mflop/s)		
		ILU	EBE	Diff	ILU	EBE	Diff	ILU	EBE	Diff
8	343000	686000	854724	+24.6%	2.30	3.25	+41.3%	298.3	263.0	−11.8%
16	1372000	2744000	3363600	+22.6%	10.66	8.53	−20.0%	257.4	394.3	+53.2%
24	3087000	6174000	7526628	+21.9%	24.50	16.88	−31.1%	252.0	445.9	+76.9%
32	5488000	10976000	13343808	+21.6%	42.38	25.58	−39.6%	259.0	521.7	+101.4%
40	8575000	17150000	20815140	+21.4%	67.47	36.66	−45.7%	254.2	567.8	+123.4%
48	12348000	24696000	29940624	+21.2%	99.65	51.51	−48.3%	247.8	581.3	+134.5%
54	16807000	33614000	40720260	+21.1%	129.14	67.79	−47.5%	260.3	600.7	+130.8%

compiler, using the flags `-O3`, `-ipo`, `-xHost`, `-inline-level=2` and `-fp-speculation=fast`.

In Table 1, we investigate the performance of the solution phase of EBE-ILU and ILU on an artificial uniformly discretized 2D grid problem with 65×65 nodes, averaged over 10000 runs. The number of rows of the element matrices (ES) was varied by taking multiple variables per geometric node. The number of non-zero values (Nnz) in the system is given. The benefit of dense operations is clear. Whenever the element matrices are large enough, the ILU(0) preconditioner is outperformed consistently.

The performance was also assessed on an artificial uniformly discretized 3D grid problem with $24 \times 24 \times 24$ nodes, averaged over 1000 runs. The size of the element matrices was varied by taking multiple variables per node. The results of those experiments are summarized in Table 2. The data demonstrates that the advantage is extended over the 2D problem. The throughput is increased by as much as 130% over ILU(0), resulting in a speedup of nearly 2 in execution time for the problems with element matrices of size 40×40, 48×48 and 54×54.

Convergence. Here, we investigate the convergence. The GNU C++ compiler v4.3.3 was used with the `-O3` flag.

BiCgSTAB is halted if the residual norm reaches the relative tolerance 10^{-10} or after 45000 iterations. We solve the second PDE model problem in [8], a 2D anisotropic electrostatic equation, for different values of the free parameter $\nu \in (0,1]$ which controls the amount of anisotropy, on a grid with 200×200 nodes. The system has 40000 unknowns and 357604 non-zeros. The coefficient matrices of the elements are given by, with $h := 1/199$:

$$A^{[e]} := \frac{1}{6h^2} \begin{bmatrix} 2+2\nu & 1-2\nu & -2+\nu & -1-\nu \\ 1-2\nu & 2+2\nu & -1-\nu & -2+\nu \\ -2+\nu & -1-\nu & 2+2\nu & 1-2\nu \\ -1-\nu & -2+\nu & 1-2\nu & 2+2\nu \end{bmatrix},$$

The data in Table 3 demonstrates the effectiveness of EBE-ILU(0) w.r.t. ILU(0). Recall that both preconditioners employ the same amount of non-zeros. ILU(0) is outperformed both in number of iterations as in the time to solve the system (which does not include the time to compute the factorization; that latter time was negligible relative to the former). The difference in solve time is smaller than the difference in number of iterations because of the 4×4 element matrices. The data in Table 1 demonstrated that for such matrices the execution time of EBE-ILU may be twice the execution time of ILU(0). Our implementation of the element-by-element matrix-vector product used by EBE-ILU was not optimal. It was up to 50% slower than the sparse matrix-vector product.

In Figure 1, we note that EBE-ILU converges more smoothly than ILU(0). That is a result of the better clustering of the eigenvalues produced by the former. We observed the smooth convergence on the other PDE model problems in [8] as well, regardless of the parameter values that were tested.

Table 3. Experiments on the convergence of EBE-ILU and ILU(0) on a 2D PDE.

ν	Time to solve system (s)				Iterations		
	ILU	EBE-ML-ILU	Diff	Speed-up	ILU	EBE-ML-ILU	Diff
0.10	14.56	9.12	−37.4%	1.60	2755	565	−79.5%
0.20	15.88	7.99	−49.7%	1.99	3025	485	−84.0%
0.30	20.21	6.43	−68.2%	3.14	3765	395	−89.5%
0.40	17.47	7.61	−82.1%	2.30	3255	475	−85.4%
0.50	†	7.80	–	–	†	435	–
0.60	14.55	6.79	−53.3%	2.14	2755	425	−84.6%
0.70	15.23	6.43	−57.8%	2.37	2895	395	−86.4%
0.80	15.97	6.61	−58.6%	2.42	2995	405	−86.5%
0.90	19.82	6.35	−68.0%	3.12	3735	385	−89.7%
1.00	24.07	4.90	−79.6%	4.91	4465	305	−93.2%

† Breakdown during factorization.

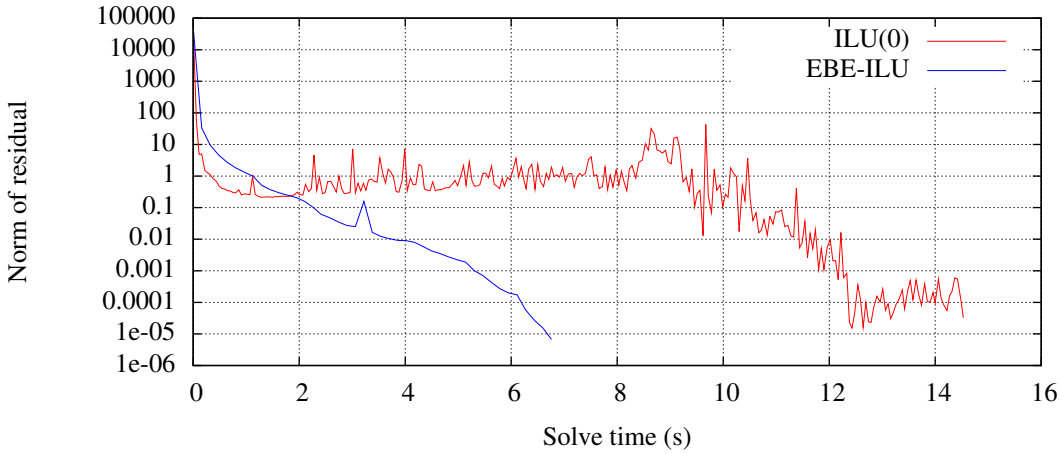

Figure 1. The convergence of EBE-ILU and ILU(0) on the model PDE with $\nu = 0.6$.

CONCLUSIONS

We proposed a multilevel block-ILU(0) preconditioner which can be constructed and applied in an element-by-element manner. The numerical factorization may be computed using BLAS3 operations. The preconditioner targets problems in which the element coefficient matrices are large or in which multiple right-hand sides are concurrently solved for.

Numerical experiments demonstrated that whenever the coefficient matrices are large, the throughput of EBE-ILU is much higher than ILU(0) and the time to solve is lower. Especially on the 3D grid problem, the solution phase of EBE-ILU was more efficient that ILU(0). The speedup was as high as 1.9. Using the same amount of non-zeros, the preconditioner manages to converge in less than a fifth of the number of iterations of ILU(0), for the model problem that was investigated. On average the speedup over ILU(0) was 2.67, with an observed maximum of about five.

REFERENCES

1. M. J. Daydé, J.-Y. L'Excellent, and N. I. M. Gould, *SIAM J. Sci. Comp.* **18**, 1767–1787 (1997).
2. M. Benzi, *J. Comp. Phys.* **182**, 418–477 (2002), ISSN 0021-9991.
3. N. Vannieuwenhoven, *An element-by-element multilevel block ILU preconditioner*, Master's thesis, K.U.Leuven (2010).
4. Y. Saad, and J. Zhang, *SIAM J. Sci. Comp.* **20**, 2103–2121 (1999).
5. Y. Saad, SPARSKIT: A basic tool kit for sparse matrix computations, Tech. Rep. RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA (1990).
6. H. A. van der Vorst, *SIAM J. Sci. Stat. Comp.* **13**, 631–644 (1992), ISSN 0196-5204.
7. GLAS, Generic Linear Algebra Software, https://www.cs.kuleuven.be/ karlm/glas (2005).
8. J. K. Kraus, *Num. Lin. Alg. Appl.* **13**, 49–70 (2006).